ML project

sharan sobhani

2022-11-26

#import libraries

```
library(tidyverse)
```

```
## -- Attaching packages --------------------------------------- tidyverse 1.3.2 --
## v ggplot2 3.3.6      v purrr   0.3.5
## v tibble  3.1.8      v dplyr   1.0.10
## v tidyr   1.2.1      v stringr 1.4.1
## v readr   2.1.2      v forcats 0.5.2

## Warning: package 'purrr' was built under R version 4.2.2

## -- Conflicts ------------------------------------------ tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
```

```
library(reshape2)
```

```
##
## Attaching package: 'reshape2'
##
## The following object is masked from 'package:tidyr':
##
##     smiths
```

```
housing<-read.csv("E:/housing.csv")
```

```
head(housing)
```

```
##   longitude latitude housing_median_age total_rooms total_bedrooms population
## 1   -122.23    37.88                 41         880            129        322
## 2   -122.22    37.86                 21        7099           1106       2401
## 3   -122.24    37.85                 52        1467            190        496
## 4   -122.25    37.85                 52        1274            235        558
## 5   -122.25    37.85                 52        1627            280        565
## 6   -122.25    37.85                 52         919            213        413
##   households median_income median_house_value ocean_proximity
## 1        126        8.3252             452600        NEAR BAY
## 2       1138        8.3014             358500        NEAR BAY
```

```
## 3        177        7.2574            352100        NEAR BAY
## 4        219        5.6431            341300        NEAR BAY
## 5        259        3.8462            342200        NEAR BAY
## 6        193        4.0368            269700        NEAR BAY
```

```
summary(housing)
```

```
##    longitude         latitude     housing_median_age  total_rooms
##  Min.   :-124.3   Min.   :32.54   Min.   : 1.00      Min.   :    2
##  1st Qu.:-121.8   1st Qu.:33.93   1st Qu.:18.00      1st Qu.: 1448
##  Median :-118.5   Median :34.26   Median :29.00      Median : 2127
##  Mean   :-119.6   Mean   :35.63   Mean   :28.64      Mean   : 2636
##  3rd Qu.:-118.0   3rd Qu.:37.71   3rd Qu.:37.00      3rd Qu.: 3148
##  Max.   :-114.3   Max.   :41.95   Max.   :52.00      Max.   :39320
##
##  total_bedrooms     population      households     median_income
##  Min.   :   1.0   Min.   :    3   Min.   :   1.0   Min.   : 0.4999
##  1st Qu.: 296.0   1st Qu.:  787   1st Qu.: 280.0   1st Qu.: 2.5634
##  Median : 435.0   Median : 1166   Median : 409.0   Median : 3.5348
##  Mean   : 537.9   Mean   : 1425   Mean   : 499.5   Mean   : 3.8707
##  3rd Qu.: 647.0   3rd Qu.: 1725   3rd Qu.: 605.0   3rd Qu.: 4.7432
##  Max.   :6445.0   Max.   :35682   Max.   :6082.0   Max.   :15.0001
##  NA's   :207
##  median_house_value ocean_proximity
##  Min.   : 14999     Length:20640
##  1st Qu.:119600     Class :character
##  Median :179700     Mode  :character
##  Mean   :206856
##  3rd Qu.:264725
##  Max.   :500001
##
```

So from that summary we can see a few things we need to do before actually running algorithms.

1)NA's in total_bedrooms need to be addressed. These must be given a value

2)We will split the ocean_proximity into binary columns. Most machine learning algorithms in R can handle categoricals in a single column, but we will cater to the lowest common denominator and do the splitting.

3)Make the total_bedrooms and total_rooms into a mean_number_bedrooms and mean_number_rooms columns as there are likely more accurate depections of the houses in a given group.

```
par(mfrow=c(2,5))
```

```
colnames(housing)
```

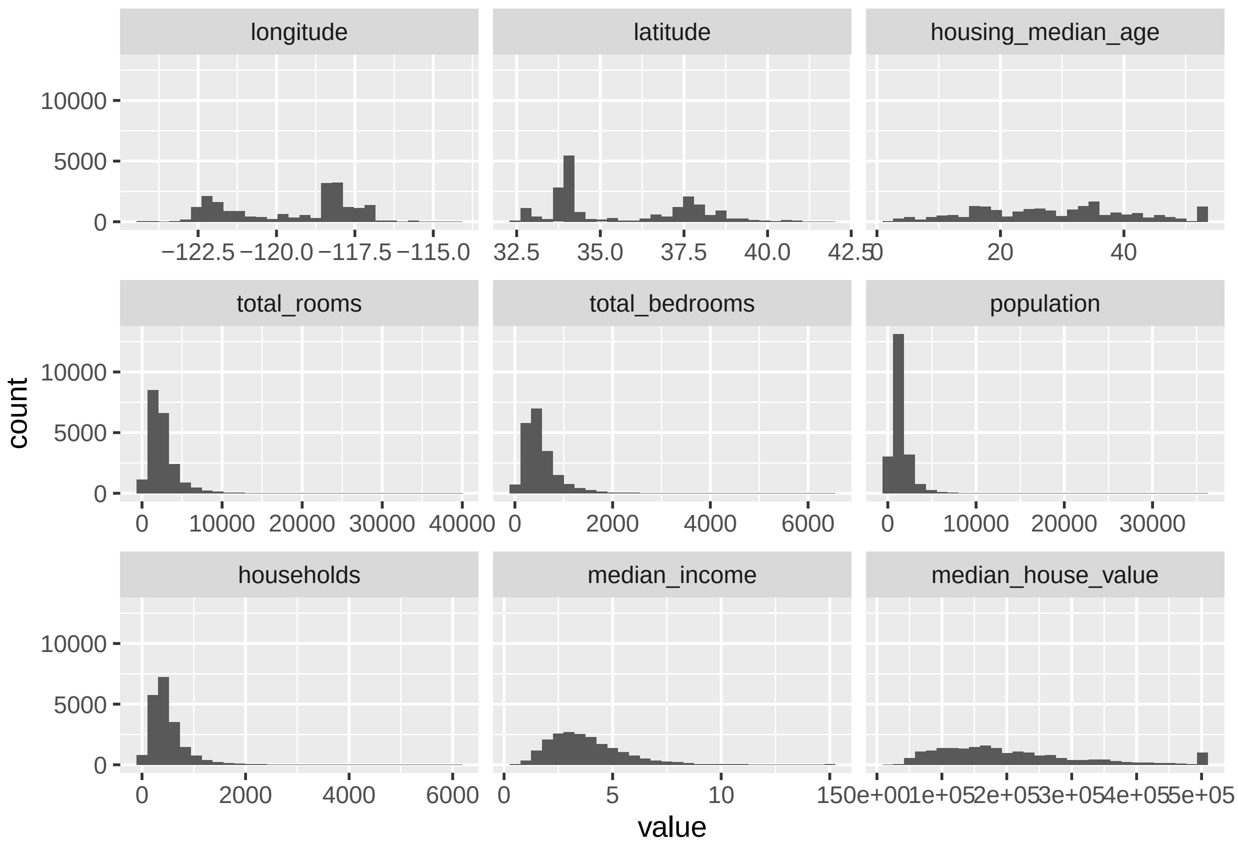
```
##  [1] "longitude"          "latitude"           "housing_median_age"
##  [4] "total_rooms"        "total_bedrooms"     "population"
##  [7] "households"         "median_income"      "median_house_value"
## [10] "ocean_proximity"
```

#lets take gender at the variables

```r
ggplot(data = melt(housing), mapping = aes(x = value)) +
    geom_histogram(bins = 30) + facet_wrap(~variable, scales = 'free_x')
```

```
## Using ocean_proximity as id variables
```

```
## Warning: Removed 207 rows containing non-finite values (stat_bin).
```



1)There are some housing blocks with old age homes in them.

2)The median house value has some weird cap applied to it causing there to be a blip at the rightmost point on the hist. There are most definitely houses in the bay area worth more than 500,000… even in the 90s when this data was collected!

3)We should standardize the scale of the data for any non-tree based methods. As some of the variables range from 0-10, while others go up to 500,000

4)We need to think about how the cap on housing prices can affect our prediction… may be worth removing the capped values and only working with the data we are confident in.

#clean the data

```r
housing$total_bedrooms[is.na(housing$total_bedrooms)] = median(housing$total_bedrooms , na.rm = TRUE)
```

Fill median for total_bedrooms which is the only column with missing values. The median is used instead of mean because it is less influenced by extreme outliers.

#fix the total coloumns- make them mean

```r
housing$mean_bedrooms = housing$total_bedrooms/housing$households
housing$mean_rooms = housing$total_rooms/housing$households

drops = c('total_bedrooms', 'total_rooms')

housing = housing[ , !(names(housing) %in% drops)]
```

```r
head(housing)
```

```
##   longitude latitude housing_median_age population households median_income
## 1   -122.23    37.88                 41        322        126        8.3252
## 2   -122.22    37.86                 21       2401       1138        8.3014
## 3   -122.24    37.85                 52        496        177        7.2574
## 4   -122.25    37.85                 52        558        219        5.6431
## 5   -122.25    37.85                 52        565        259        3.8462
## 6   -122.25    37.85                 52        413        193        4.0368
##   median_house_value ocean_proximity mean_bedrooms mean_rooms
## 1             452600        NEAR BAY     1.0238095   6.984127
## 2             358500        NEAR BAY     0.9718805   6.238137
## 3             352100        NEAR BAY     1.0734463   8.288136
## 4             341300        NEAR BAY     1.0730594   5.817352
## 5             342200        NEAR BAY     1.0810811   6.281853
## 6             269700        NEAR BAY     1.1036269   4.761658
```

Turn categoricals into booleans

1)Get a list of all the categories in the 'ocean_proximity' column

2)Make a new empty dataframe of all 0s, where each category is its own colum

3)Use a for loop to populate the appropriate columns of the dataframe

4)Drop the original column from the dataframe.

```r
categories = unique(housing$ocean_proximity)
#split the categories off
cat_housing = data.frame(ocean_proximity = housing$ocean_proximity)
```

```r
for(cat in categories){
    cat_housing[,cat] = rep(0, times= nrow(cat_housing))
}
head(cat_housing) #see the new columns on the right
```

```
##   ocean_proximity NEAR BAY <1H OCEAN INLAND NEAR OCEAN ISLAND
## 1        NEAR BAY        0         0      0          0      0
## 2        NEAR BAY        0         0      0          0      0
## 3        NEAR BAY        0         0      0          0      0
## 4        NEAR BAY        0         0      0          0      0
## 5        NEAR BAY        0         0      0          0      0
## 6        NEAR BAY        0         0      0          0      0
```

```r
for(i in 1:length(cat_housing$ocean_proximity)){
    cat = as.character(cat_housing$ocean_proximity[i])
    cat_housing[,cat][i] = 1
}

head(cat_housing)
```

```
##   ocean_proximity NEAR BAY <1H OCEAN INLAND NEAR OCEAN ISLAND
## 1        NEAR BAY        1         0      0          0      0
## 2        NEAR BAY        1         0      0          0      0
## 3        NEAR BAY        1         0      0          0      0
## 4        NEAR BAY        1         0      0          0      0
## 5        NEAR BAY        1         0      0          0      0
## 6        NEAR BAY        1         0      0          0      0
```

```r
cat_columns = names(cat_housing)
keep_columns = cat_columns[cat_columns != 'ocean_proximity']
cat_housing = select(cat_housing,one_of(keep_columns))

tail(cat_housing)
```

```
##       NEAR BAY <1H OCEAN INLAND NEAR OCEAN ISLAND
## 20635        0         0      1          0      0
## 20636        0         0      1          0      0
## 20637        0         0      1          0      0
## 20638        0         0      1          0      0
## 20639        0         0      1          0      0
## 20640        0         0      1          0      0
```

Scale the numerical variables

Note here I scale every one of the numericals except for 'median_house_value' as this is what we will be working to predict. The x values are scaled so that coefficients in things like support vector machines are given equal weight, but the y value scale doen't affect the learning algorithms in the same way (and we would just need to re-scale the predictions at the end which is another hassle).

```r
colnames(housing)
```

```
##  [1] "longitude"          "latitude"           "housing_median_age"
##  [4] "population"         "households"         "median_income"
##  [7] "median_house_value" "ocean_proximity"    "mean_bedrooms"
## [10] "mean_rooms"
```

```r
drops = c('ocean_proximity','median_house_value')
housing_num =  housing[ , !(names(housing) %in% drops)]
```

```r
head(housing_num)
```

```
##   longitude latitude housing_median_age population households median_income
## 1   -122.23    37.88                 41        322        126        8.3252
## 2   -122.22    37.86                 21       2401       1138        8.3014
```

```
## 3   -122.24    37.85                 52        496        177        7.2574
## 4   -122.25    37.85                 52        558        219        5.6431
## 5   -122.25    37.85                 52        565        259        3.8462
## 6   -122.25    37.85                 52        413        193        4.0368
##   mean_bedrooms mean_rooms
## 1     1.0238095   6.984127
## 2     0.9718805   6.238137
## 3     1.0734463   8.288136
## 4     1.0730594   5.817352
## 5     1.0810811   6.281853
## 6     1.1036269   4.761658
```

```
scaled_housing_num = scale(housing_num)
```

```
head(scaled_housing_num)
```

```
##      longitude latitude housing_median_age population households median_income
## [1,] -1.327803 1.052523          0.9821189 -0.9744050 -0.9770092    2.34470896
## [2,] -1.322812 1.043159         -0.6070042  0.8614180  1.6699206    2.33218146
## [3,] -1.332794 1.038478          1.8561366 -0.8207575 -0.8436165    1.78265622
## [4,] -1.337785 1.038478          1.8561366 -0.7660095 -0.7337637    0.93294491
## [5,] -1.337785 1.038478          1.8561366 -0.7598283 -0.6291419   -0.01288068
## [6,] -1.337785 1.038478          1.8561366 -0.8940491 -0.8017678    0.08744452
##      mean_bedrooms mean_rooms
## [1,]  -0.148510661  0.6285442
## [2,]  -0.248535936  0.3270334
## [3,]  -0.052900657  1.1555925
## [4,]  -0.053646030  0.1569623
## [5,]  -0.038194658  0.3447024
## [6,]   0.005232996 -0.2697231
```

Merge the altered numerical and categorical dataframes

```
cleaned_housing = cbind(cat_housing, scaled_housing_num, median_house_value=housing$median_house_value)

head(cleaned_housing)
```

```
##   NEAR BAY <1H OCEAN INLAND NEAR OCEAN ISLAND longitude latitude
## 1        1         0      0          0      0 -1.327803 1.052523
## 2        1         0      0          0      0 -1.322812 1.043159
## 3        1         0      0          0      0 -1.332794 1.038478
## 4        1         0      0          0      0 -1.337785 1.038478
## 5        1         0      0          0      0 -1.337785 1.038478
## 6        1         0      0          0      0 -1.337785 1.038478
##   housing_median_age population households median_income mean_bedrooms
## 1          0.9821189 -0.9744050 -0.9770092    2.34470896  -0.148510661
## 2         -0.6070042  0.8614180  1.6699206    2.33218146  -0.248535936
## 3          1.8561366 -0.8207575 -0.8436165    1.78265622  -0.052900657
## 4          1.8561366 -0.7660095 -0.7337637    0.93294491  -0.053646030
## 5          1.8561366 -0.7598283 -0.6291419   -0.01288068  -0.038194658
## 6          1.8561366 -0.8940491 -0.8017678    0.08744452   0.005232996
##   mean_rooms median_house_value
```

```
## 1  0.6285442             452600
## 2  0.3270334             358500
## 3  1.1555925             352100
## 4  0.1569623             341300
## 5  0.3447024             342200
## 6 -0.2697231             269700
```

Create a test set of data

```
set.seed(1738) # Set a random seed so that same sample can be reproduced in future runs

sample = sample.int(n = nrow(cleaned_housing), size = floor(.8*nrow(cleaned_housing)), replace = F)
train = cleaned_housing[sample, ] #just the samples
test  = cleaned_housing[-sample, ] #everything but the samples
```

Note that the train data below has all the columns we want, and also that the index is jumbled (so we did take a random sample). The second check makes sure that the length of the train and test dataframes equals the length of the dataframe they were split from, which shows we didn't lose data or make any up by accident!

```
head(train)
```

```
##       NEAR BAY <1H OCEAN INLAND NEAR OCEAN ISLAND  longitude   latitude
## 15797        1         0      0          0      0 -1.4226356  0.9963418
## 11425        0         1      0          0      0  0.7984423 -0.8997679
## 9208         0         0      1          0      0 -0.1399007  0.6873461
## 8778         0         1      0          0      0  0.6287420 -0.8623139
## 18375        0         1      0          0      0 -1.1431292  0.7482089
## 19571        0         0      1          0      0 -0.6889312  0.9167520
##       housing_median_age population   households median_income mean_bedrooms
## 15797          1.8561366  0.7360275  1.254049198    -0.7002610   -0.09402939
## 11425         -0.2891796  0.1938460 -0.006642643     1.0257957   -0.26800948
## 9208          -1.6399342 -0.8940491 -0.924698320    -0.2609040    0.09587571
## 8778           0.5053819 -0.0454556  0.079670283    -0.3281734   -0.07443967
## 18375         -0.3686357  0.4508082  0.398766559     1.0631150   -0.14119530
## 19571          1.1410312 -0.4975679 -0.438207278    -1.0654005   -0.06673316
##        mean_rooms median_house_value
## 15797 -0.71339265             250000
## 11425  0.46743428             286100
## 9208   0.09790468              80700
## 8778  -0.44792541             254700
## 18375  0.49424219             271400
## 19571 -0.29391335              81500
```

```
nrow(train) + nrow(test) == nrow(cleaned_housing)
```

```
## [1] TRUE
```

Test some predictive models simple linear model using 3 of the avaliable predictors. Median income, total rooms and population. This serves as an entry point to introduce the topic of cross validation and a basic model

So here we do cross validation to test the model using the training data itself. Our K is 5, what this means is that the training data is split into 5 equal portions. One of the 5 folds is put to the side (as a mini test data set) and then the model is trained using the other 4 portions. After that the predictions are made on the folds that was withheld, and the process is repeated for each of the 5 folds and the average predictions produced from the iterations of the model is taken. This gives us a rough understanding of how well the model predicts on external data!

```
library('boot')
```

```
glm_house = glm(median_house_value~median_income+mean_rooms+population, data=cleaned_housing)
k_fold_cv_error = cv.glm(cleaned_housing , glm_house, K=5)
```

```
k_fold_cv_error$delta
```

```
## [1] 6993810248 6983982760
```

The first component is the raw cross-validation estimate of prediction error. The second component is the adjusted cross-validation estimate.

```
glm_cv_rmse = sqrt(k_fold_cv_error$delta)[1]
glm_cv_rmse #off by about $83,000... it is a start
```

```
## [1] 83629
```

```
glm_house$coefficients
```

```
##   (Intercept) median_income    mean_rooms    population
##    206855.817     82608.959     -9755.442     -3948.293
```

```
library("randomForest")
```

```
## Warning: package 'randomForest' was built under R version 4.2.2
```

```
## randomForest 4.7-1.1
```

```
## Type rfNews() to see new features/changes/bug fixes.
```

```
##
## Attaching package: 'randomForest'
```

```
## The following object is masked from 'package:dplyr':
##
##     combine
```

```
## The following object is masked from 'package:ggplot2':
##
##     margin
```

```r
names(train)
```

```
##  [1] "NEAR BAY"           "<1H OCEAN"          "INLAND"
##  [4] "NEAR OCEAN"         "ISLAND"             "longitude"
##  [7] "latitude"           "housing_median_age" "population"
## [10] "households"         "median_income"      "mean_bedrooms"
## [13] "mean_rooms"         "median_house_value"
```

```r
set.seed(1738)

train_y = train[,'median_house_value']
train_x = train[, names(train) !='median_house_value']

head(train_y)
```

```
## [1] 250000 286100  80700 254700 271400  81500
```

```r
head(train_x)
```

```
##       NEAR BAY <1H OCEAN INLAND NEAR OCEAN ISLAND  longitude   latitude
## 15797        1         0      0          0      0 -1.4226356  0.9963418
## 11425        0         1      0          0      0  0.7984423 -0.8997679
## 9208         0         0      1          0      0 -0.1399007  0.6873461
## 8778         0         1      0          0      0  0.6287420 -0.8623139
## 18375        0         1      0          0      0 -1.1431292  0.7482089
## 19571        0         0      1          0      0 -0.6889312  0.9167520
##       housing_median_age population   households median_income mean_bedrooms
## 15797          1.8561366  0.7360275  1.254049198    -0.7002610   -0.09402939
## 11425         -0.2891796  0.1938460 -0.006642643     1.0257957   -0.26800948
## 9208          -1.6399342 -0.8940491 -0.924698320    -0.2609040    0.09587571
## 8778           0.5053819 -0.0454556  0.079670283    -0.3281734   -0.07443967
## 18375         -0.3686357  0.4508082  0.398766559     1.0631150   -0.14119530
## 19571          1.1410312 -0.4975679 -0.438207278    -1.0654005   -0.06673316
##        mean_rooms
## 15797 -0.71339265
## 11425  0.46743428
## 9208   0.09790468
## 8778  -0.44792541
## 18375  0.49424219
## 19571 -0.29391335
```

```r
#rf_model = randomForest(median_house_value~. , data = train, ntree =500, importance = TRUE)
rf_model = randomForest(train_x, y = train_y , ntree = 500, importance = TRUE)
```

```r
names(rf_model) #these are all the different things you can call from the model.
```

```
##  [1] "call"            "type"            "predicted"       "mse"
##  [5] "rsq"             "oob.times"       "importance"      "importanceSD"
##  [9] "localImportance" "proximity"       "ntree"           "mtry"
## [13] "forest"          "coefs"           "y"               "test"
## [17] "inbag"
```

```
rf_model$importance
```

```
##                       %IncMSE IncNodePurity
## NEAR BAY            486443080  1.312625e+12
## <1H OCEAN          1621072507  4.289632e+12
## INLAND             4045427703  3.068877e+13
## NEAR OCEAN          539828604  2.299546e+12
## ISLAND                1524086  6.496858e+10
## longitude          6897270047  2.572075e+13
## latitude           5710904041  2.255767e+13
## housing_median_age 1082226582  9.661389e+12
## population         1066423080  7.341037e+12
## households         1193112832  7.923472e+12
## median_income      8486026742  7.325002e+13
## mean_bedrooms       402648032  7.555405e+12
## mean_rooms         1820857979  2.120577e+13
```

The out-of-bag (oob) error estimate In random forests, there is no need for cross-validation or a separate test set to get an unbiased estimate of the test set error. It is estimated internally, during the run, as follows:

Each tree is constructed using a different bootstrap sample from the original data. About one-third of the cases are left out of the bootstrap sample and not used in the construction of the kth tree.

```
oob_prediction = predict(rf_model) #leaving out a data source forces OOB predictions
```

```
#you may have noticed that this is avaliable using the $mse in the model options.
#but this way we learn stuff!
train_mse = mean(as.numeric((oob_prediction - train_y)^2))
oob_rmse = sqrt(train_mse)
oob_rmse
```

```
## [1] 49126.22
```

So even using a random forest of only 1000 decision trees we are able to predict the median price of a house in a given district to within $49,000 of the actual median house price. This can serve as our bechmark moving forward and trying other models.

How well does the model predict on the test data?

```
test_y = test[,'median_house_value']
test_x = test[, names(test) !='median_house_value']


y_pred = predict(rf_model , test_x)
test_mse = mean(((y_pred - test_y)^2))
test_rmse = sqrt(test_mse)
test_rmse
```

```
## [1] 47625.57
```

Well that looks great! Our model scored roughly the same on the training and testing data, suggesting that it is not overfit and that it makes good predictions.